

ICASE

FINDING THE MINIMUM DISTANCE BETWEEN TWO CONVEX POLYGONS

Jacob T. Schwartz

Report No. 81-22

July 30, 1981

INSTITUTE FOR COMPUTER APPLICATIONS IN SCIENCE AND ENGINEERING
NASA Langley Research Center, Hampton, Virginia 23665

Operated by the

UNIVERSITIES SPACE



RESEARCH ASSOCIATION

(NASA-CR-185794) FINDING THE MINIMUM
DISTANCE BETWEEN TWO CONVEX POLYGONS
(ICASE) 10 p

N89-71370

Unclas
00/64 0224377

FINDING THE MINIMUM DISTANCE BETWEEN TWO CONVEX POLYGONS

Jacob T. Schwartz
*Courant Institute of Mathematical Sciences,
New York University*

ABSTRACT

An $O(\log^2 N)$ algorithm for determining the minimum distance between two polygons is given, together with a technique for improving its average-case performance to $O(\log N)$.

This report was prepared as a result of work performed under NASA Contracts No. NAS1-14472 and NAS1-15810 while the author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23665.

1. Introduction

Suppose that a provisional path of coordinated motion has been planned for two rigid polyhedral bodies B_1 and B_2 in 3-space. Then there will exist functions $R_1(t)$, $R_2(t)$, $x_1(t)$, $x_2(t)$ of a parameter t designating time such that the set of points occupied by B_j at time t is $R_j(t)B_j + x_j(t)$. To verify the validity of the proposed motion, one can proceed as follows: find the minimum distance δ between the convex sets $R_1(t)B_1 + x_1(t)$ and $R_2(t)B_2 + x_2(t)$ for $t=0$. Let L be the diameter of the set B_2 , and from the known form of the functions R_1 , R_2 , x_1 , and x_2 find an ϵ sufficiently small so that

$$(|R_1^{-1}(t)R_2(t) - R_1^{-1}(0)R_2(0)| \cdot L + |x_1(t) + x_2(t) - x_1(0) - x_2(0)| < \delta,$$

for all $0 \leq t \leq \epsilon$. Then a collision between the moving bodies is impossible for this range of t ; hence we can advance t from 0 to ϵ , and repeat this step. When successive steps of this kind have brought us from $t=0$ to some final value t^* , we can be sure that the planned path is collision-free.

To use this technique effectively, we need a fast algorithm for estimating the minimum distance between two polyhedra. The present note will address the problem of finding this distance, but only under two drastic simplifying assumptions, namely

- (i) B_1 and B_2 are assumed to be convex; and
- (ii) B_1 and B_2 are assumed to be two dimensional.

Assuming that B_1 and B_2 have a total of N vertices and are described by clockwise bounding segment lists of the standard kind, an $O(\log^2 N)$ algorithm for determining the minimum distance between B_1 and B_2 will be given. The related problem of finding the minimum distance between a

variable point x and a fixed convex body B is considered in [Shamos], where an $O(\log N)$ algorithm is given.

2. The algorithm

Hence let B_1 and B_2 be convex polygons. Write

$$B_1 \pm B_2 = \{x \pm y : x \in B_1, y \in B_2\}$$

and

$$-B = \{-x : x \in B\},$$

so that $B_1 + B_2$ is the so-called Minkowski Sum of B_1 and B_2 , and $B_1 - B_2 = B_1 + (-B_2)$. Our problem is to estimate the distance between the point $x=0$ and the set $B_1 - B_2$; replacing B_2 by $-B_2$, it becomes that of estimating the distance from the origin to the convex set $B_1 + B_2$.

A fast $O(N)$ procedure for finding $B_1 + B_2$ given B_1 and B_2 is described in [Najfeld], and is as follows:

(a) The sides of B_1 and B_2 are available as circular lists arranged in increasing order of the angle θ that each side makes with the x axis. Merge these two lists into a single similarly ordered list L .

(b) If a side S of B_2 (resp. B_1) lies between two successive sides S', S'' of B_1 (resp. B_2) in this list, let c be the corner at which S and S' meet. Then $S+c$ is a side of $B_1 + B_2$, and L lists these sides of $B_1 + B_2$ in their standard circular order. We will say in what follows that the side $S+c$ of $B_1 + B_2$ comes from S (which is a side either of B_1 or of B_2), and that c is the corner of B_1 (resp. B_2) that matches the side S of B_2 (resp. B_1).

Our algorithm will apply this construction, but to attain $O(\log^2 N)$ performance will avoid forming the full list L . We proceed as follows.

(i) The sides of any convex polygon B can be oriented so that the interior of B lies to their left. Oriented in this way, the sides of B fall into two (circularly) contiguous groups, one consisting of (vertically) ascending, the other of descending edges. It is clear that the ascending (resp. descending) edges of $B_1 + B_2$ come from the separate ascending (resp. descending) edges of B_1 and of B_2 .

(ii) Let a point x in the plane be given; we wish to determine whether it belongs to $B_1 + B_2$, and if not, to find its minimum distance to $B_1 + B_2$. First determine the highest (resp. lowest) corner H_j L_j of each B_j ; this can be done in time $O(\log N)$ by binary search of the sides of B_j . Then the highest and lowest corners of $B_1 + B_2$ are $H_1 + H_2$ and $L_1 + L_2$ respectively. If x does not lie in the vertical range bracketed by these two points, it is definitely not in $B_1 + B_2$. If x does lie in this vertical range, then a horizontal line drawn through x will intersect exactly two sides of $B_1 + B_2$, one an ascending, the other a descending side, and x belongs to $B_1 + B_2$ if and only if it lies between these two sides. To find these two intersections, we can start at the lowest corner $L_1 + L_2$ of $B_1 + B_2$, and perform a binary search of its ascending (resp. descending) sides. This can be done without actually forming the full collection of its sides, using a technique which we will now explain.

(iii) Let the ascending sides of $B_1 + B_2$ be enumerated, in bottom-to-top order, as S_1, S_2, \dots, S_n . At any moment during a binary search of these sides, we will be examining two of these sides S_i, S_k , and will need to examine a side S_j lying between them. We can suppose that the

side S' of B_1 (resp. B_2) from which each side S_j under examination comes is known, and that the corner c of B_2 (resp. B_1) matching S' is also known. For our binary search to retain logarithmic efficiency, we must be able to locate a side S_k between S_i and S_j such that k is at least (resp. most) a fixed fraction α (resp. $1-\alpha$) of the distance from i to j . This can be done as follows. Let S_i come from a side S' (of B_1 or B_2) numbered i' , matched by a corner c'' (of B_2 or B_1) whose entering edge is numbered i'' , and define S'' , j' and j'' , similarly from S_j . If S' and S'' are sides of the same polygon (B_1 or B_2), put

$$\Delta' = j' - i', \quad \Delta'' = j'' - i''.$$

On the other hand, if S' and S'' are sides of different polygons, put

$$\Delta' = j'' - i', \quad \Delta'' = j' - i''.$$

In order to avoid detailed enumeration of tediously many cases, we will suppose that S' and S'' are sides of different polygons; the treatment of the cases thereby ignored and of this case are similar.

If $\Delta' \geq \Delta''$, advance from side i' (of the polygon having S' as a side) halfway toward side j'' of this polygon. Let the side in this intermediate position be T , let its index be m , and find its matching corner c . Then $T+c$ is a side of B_1+B_2 ; its index as a side of B_1+B_2 exceeds i by at least $\Delta'/2$ and by at most $\Delta'/2+\Delta''$. Hence m lies at least $1/4$ and at most $3/4$ of the way from i to j .

Similarly, if $\Delta'' > \Delta'$, advance from side i'' (of the polygon having c'' as a corner) halfway toward side i' of this polygon. Let the side in this intermediate position be T , its index be m , and its matching corner

be c . Then again $T+c$ is a side of B_1+B_2 whose index is at least $1/4$ and at most $3/4$ of the way from i to j .

Locating the corner matching a given side can be done in time $O(\log N)$, so overall the binary search we have just described requires $O(\log^2 N)$ time.

(iv) The binary search will locate the two points of intersection of the horizontal line through x with the boundary of B_1+B_2 . If x lies between these points it is interior to B_1+B_2 , and we are finished. Otherwise x lies to the right or to the left of one of them. Suppose, for the sake of definiteness, that x lies to the right of B_1+B_2 , or, if x lies above (resp. below) the topmost (resp. bottom most) point of B_1+B_2 , that it lies to the left of this point. Then the point of B_1+B_2 lying closest to x lies on one of the ascending edges forming the left-hand part Q of the boundary of B_1+B_2 . We now begin to search for this edge. We start this search from an edge of B_1+B_2 visible from x . Such an edge is available in all cases, since if x lies above (resp. below) B_1+B_2 we have only to take the topmost (resp. bottom most) edge of Q .

(v) To find the edge S^* of B_1+B_2 containing the point Z closest to x , we start with an edge S visible from x and draw a line from x to the initial corner c_1 of S (note again that the edges of Q are oriented and point downward). Let c_2 be the other corner of S . If the angle xc_1c_2 is acute, then N lies on Q below c_1 ; if obtuse, then at or above c_1 . This observation enables the edge S containing Z to be located by binary search. As previously, this binary search procedure will run in time $O(\log^2 N)$. Suppose now that S contains Z . Then if xc_1c_2 is acute but xc_2c_1 is obtuse Z is

c_2 ; if $\angle x c_1 c_2$ is obtuse then Z is c_1 ; and otherwise Z is the foot of the perpendicular from x to S .

3. A technique for accelerating the expected speed of location of a point on a divided real axis

Like many other geometric algorithms, the algorithm sketched in the preceding pages makes repeated use of the following computational step:

Given a fixed increasing sequence of real numbers

x_1, \dots, x_n , and a point x , locate the interval (x_i, x_{i+1}) in which x lies.

The normal technique for accomplishing this is simply to perform a binary search, which requires time $O(\log n)$.

We will now sketch an alternative approach which has the same worst case behavior, but (if the points x_j are randomly distributed) will reduce the expected time needed to locate the desired interval to $O(1)$. This is simply to keep an auxiliary table T consisting of n/α locations. To set up T , we divide the full range (x_1, x_n) from the minimum to the maximum of the x_i into αn equal subintervals I , each of which corresponds to an entry E of T ; E then stores the indices of the largest and smallest x_j belonging to I . To find the interval (x_i, x_{i+1}) containing a given x we simply calculate the entry E of T corresponding to x , and perform a binary search in the subrange (x_j, x_k) of (x_1, x_n) indicated by E .

To analyze the expected performance of this scheme, we can reason as follows. The number of x_j expected to fall into each of the subranges I into which we divide the full range (x_1, x_n) is α , so that,

assuming that α is small, the probability p_j that j items actually fall into I will be Poissonian with expectation α , i.e.

$p_j = e^{-\alpha} \alpha^j / j!$. If we enter an interval containing j items to do a binary search, $O(1 + \log j)$ time will be required for the search. Thus the expected searching time is $O(1) + O(e^{-\alpha} \sum_{j \geq 2} \alpha^j \log j / j!)$. The following is a table of this last function:..

<u>Value of α</u>	<u>Value of $e^{-\alpha} \sum_{j \geq 2} \alpha^j \log j / j!$</u>
1	.22
2	.57
4	1.20

This technique can be used in the 'find matching corner' step of the closest point algorithm sketched earlier, and, assuming a random distribution of angles parallel to the sides of the polygons involved, will reduce the expected time needed to find this corner to $O(1)$; thus the expected time required for the whole algorithm is $O(\log N)$ rather than $O(\log^2 N)$.

Bibliography

I. Najfeld [1978] Analytic design of compensators and computational geometry. Ph.D. Thesis, Brown University, Providence, RI.

M. Shamos [1975] Problems in computational geometry (first revision), Informally distributed lecture notes, Carnegie-Mellon University, Pittsburgh, PA.